**Naval Research Laboratory**

Washington, DC 20375-5000

NRL Memorandum Report 6359

# Alternative Trace Axioms for the WHILE Construct

C. B. CROSS

*Formal Methods Section*
*Information Technology Division*



October 21, 1988

SECURITY CLASSIFICATION OF THIS PAGE

## REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

| | |
|---|---|
| 1a REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED | 1b RESTRICTIVE MARKINGS |
| 2a SECURITY CLASSIFICATION AUTHORITY | 3 DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for public release; distribution unlimited. |
| 2b DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4 PERFORMING ORGANIZATION REPORT NUMBER(S)<br>NRL Memorandum Report 6359 | 5 MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a NAME OF PERFORMING ORGANIZATION<br>Naval Research Laboratory<br>6b OFFICE SYMBOL (If applicable) | 7a NAME OF MONITORING ORGANIZATION |
| 6c ADDRESS (City, State, and ZIP Code)<br>Washington, DC 20375-5000 | 7b ADDRESS (City, State, and ZIP Code) |
| 8a NAME OF FUNDING/SPONSORING ORGANIZATION<br>8b OFFICE SYMBOL (If applicable) | 9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
| 8c ADDRESS (City, State, and ZIP Code) | 10 SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO |

11 TITLE (Include Security Classification)
Alternative Trace Axioms for the WHILE Construct

12 PERSONAL AUTHOR(S)
Cross, C.B.

| 13a TYPE OF REPORT | 13b TIME COVERED FROM 11/87 TO 6/88 | 14 DATE OF REPORT (Year, Month, Day) | 15 PAGE COUNT |
|---|---|---|---|
| | | 1988 October 21 | 14 |

16 SUPPLEMENTARY NOTATION

| 17 COSATI CODES | | | 18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Logic, Verification, Program semantics, Trace specification |
| | | | |
| | | | |

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

In NRL Report 9033 John McLean presents a programming language semantics, the extended trace language, based on the trace specification language. The simple programming language discussed in NRL Report 9033 contains the WHILE construct, and McLean gives this construct a natural and correct recursive treatment. In this report I show that it is possible to use the extended trace language to give the WHILE construct two other quite different semantic treatments. One of these is based on the Hoare-style semantics for WHILE; the second is an alternative to the recursive axiom that could be used in cases where the verifier can discern at what point a given loop will terminate. It is significant that when using the extended trace language a verifier of software can choose from several different but equivalent semantic treatments of WHILE. The ability to choose an axiom for WHILE that fits the problem at hand makes the extented trace language an attractive software verification formalism.

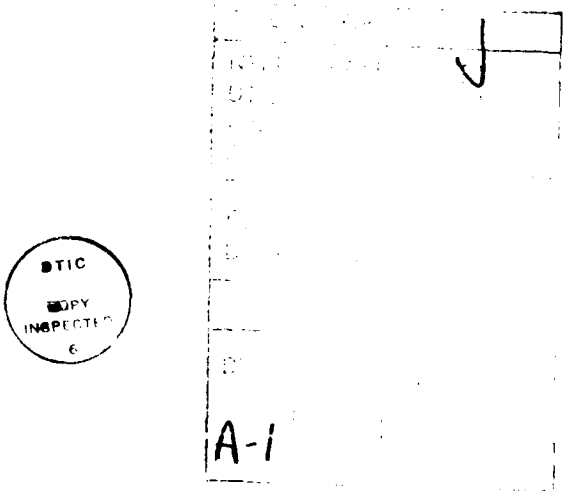
| 20 DISTRIBUTION/AVAILABILITY OF ABSTRACT | 21 ABSTRACT SECURITY CLASSIFICATION |
|---|---|
| ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS | UNCLASSIFIED |
| 22a NAME OF RESPONSIBLE INDIVIDUAL<br>Charles B. Cross | 22b TELEPHONE (Include Area Code) / 22c OFFICE SYMBOL |

DD Form 1473, JUN 86 Previous editions are obsolete SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

## CONTENTS





# Alternative Trace Axioms for the WHILE Construct

ONE FAMILIAR AND TRIVIAL FACT about formal logic is that an inference may be easy to derive in one proof system but difficult to derive in a second, equivalent system. It is therefore not surprising that some programs are more difficult to prove correct using Hoare logic than, for example, dynamic logic, and vice-versa. It would be good to have a single logic of programs that made correctness proofs easy in all cases, but this is too much to hope for, since any finite, reasonably rich axiomatic system will have theorems that can only be derived by means of lengthy proofs. Still, one can hope to make a certain amount of progress toward the desired end before encountering the point of diminishing marginal returns. The usual area of difficulty in any verification proof is proving that repetition constructs (such as loops and recursion) behave as they were intended. In view of this, the present report focuses on different ways to handle the archetypal loop construct: **while**.

In [6] John McLean presents a programming language semantics, the *extended trace language*, based on the trace specification language described in [5]. The simple programming language discussed in [6] contains the **while** construct, and McLean gives this construct a natural and correct recursive treatment. In this report I will show that it is possible to use the extended trace language to give the **while** construct two other quite different semantic treatments. One of these is based on the Hoare-style semantics for **while**; the second is an alternative to the recursive axiom that could be used in cases where the verifier can discern at what point a given loop will terminate. It is significant that when using the extended trace language a verifier of software can choose from several different but equivalent semantic treatments of **while**. The ability to choose an axiom for **while** that fits the problem at hand makes the extended trace language an attractive software verification formalism.

## 1 The trace specification language

The trace language provides for the specification of software modules in terms of the effects, such as return-values, that the user sees when executing a sequence of procedure and function calls. These sequences are called *traces*.

A trace specification consists of a *syntax* section and a *semantics* section. The syntax section states the name and parameter types of each of the module's procedures and the name, parameter types, and return-value type of each of the module's function calls. The semantics section contains axioms formalized in a many-sorted language of first-order logic with identity, with one set of variables $R, R_1, R_2, \ldots S, S_1, S_2, \ldots T, T_1, T_2, \ldots$ to be understood as ranging over traces. In addition to the usual logical connectives there is an interpreted binary function symbol (.), which serves as a notation for concatenating trace terms. If $X$ is a trace variable, the empty trace $e$, a procedure call, or a function call, then $X$ is a well-formed trace term; if $X$ and $Y$ are well-formed trace terms, then $(X.Y)$ is a well-formed trace term. Nothing else is a well-formed trace term. A function (procedure) call is a function (procedure) name followed by the requisite number of parameters of appropriate types. In place of a formal axiom of associativity for concatenation I adopt the convention of dropping the parentheses around the subterms of a trace term.

The axioms that appear in the semantics section of a trace specification state or entail information about which traces are *legal* and about the values returned by legal traces that end with

Manuscript approved August 19, 1988.

function calls. The legality predicate and the value function are usually formalized using the unary predicate symbol $L$ and the unary function symbol $V$, respectively. One additional and very handy piece of notation is *trace equivalence* ($\equiv$), defined as follows:[1]

$$S \equiv T =_{df} \forall R\,[(L(S.R) \leftrightarrow L(T.R)) \wedge (R \neq e \rightarrow (\exists x\, V(S.R) = x \rightarrow V(S.R) = V(T.R)))]$$

In other words, two traces are equivalent just in case they agree on (i) present and future legality, and (ii) all future return values. Intuitively, two traces are equivalent provided that they place the module in the same "state" as far as the user can tell.

## 2 The extended trace specification language

McLean's extended trace semantics is defined on a simple programming language that permits variable assignment, sequencing, **while do**, and **if then else**. The extended trace language itself is very much like the trace specification language, except that in addition to procedure call variables $R$ and $T$, with and without subscripts, we have program statement variables $S$, with and without subscripts. In addition, the value function $V$ takes two arguments instead of one. The first argument is a trace expression, the second a program variable or Boolean expression. McLean's program semantics consists of a complete set of axioms and rules for first-order logic with identity and functions, together with the following additional axioms:

1. $V(S, c) = c$, for constant $c$.
2. $V(e, x) = 0$, for any integer variable $x$.[2]
3. $V(S.a := t, a) = V(S, t)$, for term $t$.
4. $V(S, \sigma(a, b)) = \sigma(V(S, a), V(S, b))$, where $\sigma$ is an arithmetical operation.
5. $V(S, \sigma(p, q)) \leftrightarrow \sigma(V(S, p), V(S, q))$, where $\sigma$ is a Boolean operation.[3]
6. $V(S.a := x, b) = V(S, b)$, unless (i) $b$ is an expression containing $a$, (ii) $b$ is an array variable whose index is an expression containing $a$, or (iii) both $a$ and $b$ are array variables to the same array.
7. $V(S, \phi) = V(S, \psi) \rightarrow V(S, a[\phi]) = V(S, a[\psi])$, where $a$ is an array.
8. $V(S, \phi) \neq V(S, \psi) \rightarrow V(S.a[\phi] := t, a[\psi]) = V(S, a[\psi])$, where $a$ is an array.
9. $V(S, \theta) \rightarrow V(S.\,\textbf{if}\ \theta\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\,, x) = V(S.S_1, x)$, where $x$ is a program variable or Boolean expression and where '$=$' is understood as meaning '$\leftrightarrow$' in the latter case.
10. $\neg V(S, \theta) \rightarrow V(S.\,\textbf{if}\ \theta\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\,, x) = V(S.S_2, x)$, where $x$ is a program variable or Boolean expression and where '$=$' is understood as meaning '$\leftrightarrow$' in the latter case.
11. $(\mathbf{acc}(S, S_0, T) \wedge \neg V(T, \theta)) \rightarrow V(S.\,\textbf{while}\ \theta\ \textbf{do}\ S_0\ \textbf{od}\,, x) = V(S.\,\textbf{if}\ \theta\ \textbf{then while}\ \theta\ \textbf{do}\ S_0\ \textbf{od fi}\,.x)$, where $x$ is a program variable or Boolean expression and with '$=$' understood as meaning '$\leftrightarrow$' in the latter case.
12. $V(S.\textbf{skip}, t) = V(S, t)$, for term $t$.

[1] [6], p. 4.

[2] This axiom states that all numerical program variables are initialized to 0.

[3] Note that if $p$ is a formula or Boolean variable, then $V(S, p)$ is a formula, not a term.

The acc predicate, which is useful for proving the correctness of programs, is defined by the following equivalence:

$$\text{acc}(T, S, R) \leftrightarrow \exists n\, R = T.S^n,$$

where $S^n$ is a function that maps a trace and an integer to a trace and is defined by the following axioms:

$$S^0 = e$$
$$S^{n+1} = S^n.S$$

So, a trace $R$ is $S$-accessible to a trace $T$ provided that $R$ is the result of appending a finite number of occurrences of $S$ onto $T$. A related predicate that will come in handy later on is ext, defined as follows:

$$\text{ext}(\phi, T, S, R) \leftrightarrow [\text{acc}(T, S, R) \wedge \forall X [[\text{acc}(T, S, X) \wedge \text{acc}(X, S, R)] \rightarrow V(X, \phi)]]$$

In other words, $R$ is an $S$-accessible $\phi$-extension of $T$ provided that $R$ is $S$-accessible to $T$ and $\phi$ is uniformly true over every prefix of $R$ having the form $T.S^i$.

In the programming language we are considering, a program fails to terminate only if one or more of its **while** loops fails to terminate. So to prove that a program halts we need only prove that its loops terminate. In general this will involve considering a trace of the form

$$R\,.\,\textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}$$

and proving

$$\exists T\,(\text{acc}(R, S, T) \wedge \neg V(T, \phi)).$$

Unfortunately, our use of first-order logic greatly limits our ability to prove that programs terminate. For example, let P be an extension of first-order logic containing the axioms of Peano arithmetic and the axioms of the extended trace language, and consider the following program:

$$x := z\,.\,\textbf{while}\ x > 0\ \textbf{do}\ x := x - 1\ \textbf{od}$$

In order to prove that this program terminates for values of $z$ greater than 0, we need to prove that P implies the following formula:

$$\forall z\,(z > 0 \rightarrow \exists T\,(\text{acc}(x := z, x := x - 1, T) \wedge \neg V(T, x > 0))).$$

But if $\mathcal{M}$ is a *nonstandard* model of arithmetic that satisfies the axioms of the extended trace language, then $\mathcal{M}$ cannot satisfy $\forall z\,(z > 0 \rightarrow \exists T\,(\text{acc}(x := z, x := x - 1, T) \wedge \neg V(T, x > 0)))$, since M cannot satisfy $\exists T\,(\text{acc}(x := z, x := x - 1, T) \wedge \neg V(T, x > 0))$ for nonstandard integer values of $z$. Thus in many cases, some of them very simple cases, it will be impossible to prove termination for a program that clearly does terminate. In [1] Apt makes this same point regarding Hoare logic, and so, as with Hoare logic, the extended trace language is more useful for proving weak correctness than for proving strong correctness.[4]

We will refer to Axiom 11 as the *recursive axiom*, since it associates a **while** statement with an **if-then-else** statement that calls that same **while** statement again.

[4] A program is *weakly* correct just in case it is correct *if* it terminates; a program is *strongly* correct just in case it is weakly correct and also terminates.

# 3 The invariance while axiom

## 3.1 Hoare logic

The semantics of **while** in Hoare's logic, by contrast, depends on the idea of an *invariant*. The standard **while** rule in Hoare logic is as follows:

$$\frac{\{p \wedge e\}\, S\, \{p\}}{\{p\}\ \textbf{while}\ e\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg e\}}$$

Proving a program $S_1$ ; **while** $e$ **do** $S_2$ **od** partially correct with respect to a precondition $p$ and postcondition $q$ normally involves showing that $S_1$ *establishes* an invariant $r$ that $S_2$ *preserves*. This means that in order to prove $\{p\}\, S_1$; **while** $e$ **do** $S_2$ **od** $\{q\}$ we must prove the following, for some cleverly chosen $r$: $\{p\}\, S_1, \{r\}$, $\{r \wedge e\}\, S_2\, \{r\}$, $(r \wedge \neg e) \rightarrow q$. From the second of these $\{r\}$ **while** $e$ **do** $S_2$ **od** $\{r \wedge \neg e\}$ follows by Hoare's **while** rule, and this, together with $\{p\}\, S_1, \{r\}$ implies by the Composition Rule[5] $\{p\}\, S_1$ **while** $e$ **do** $S_2$ **od** $\{r \wedge \neg e\}$. This last formula, together with $(r \wedge \neg e) \rightarrow q$ implies the target formula by the Consequence Rule:[6] $\{p\}\, S_1$ ; **while** $e$ **do** $S_2$ **od** $\{q\}$.[7]

Consider, for example, the following program, which computes the factorial function for input $x$:

```
a := 1;
b := x;
while b ≠ 0 do
    a := ab;
    b := b − 1 od;
end
```

In order to prove this program (weakly) correct, we need to give a Hoare logic proof of the following:

**Proposition 1** $\{x \geq 0\}\ a := 1; b := x;$ **while** $b \neq 0$ **do** $a := ab; b := b - 1$ **od** $\{a = x!\}$.

**Proof:** To prove this, according to the procedure described above, it suffices to prove each of the following:

**(1)** $\{x \geq 0\}\ a := 1; b := x\ \{r\}$
**(2)** $\{r \wedge b \neq 0\}\ a := ab; b := b - 1\ \{r\}$
**(3)** $(r \wedge b = 0) \rightarrow a = x!$

for suitably chosen $r$. Let $r$ be the formula $a = x!/b!$. Obviously (3) holds, since $0! = 1$, so consider (1): by the Assignment Axiom[8] and the Consequence Rule it follows that

$$\{x \geq 0\}\ a := 1\ \{x \geq 0 \wedge a = x!/x!\}$$

[5] [1], p. 433.
[6] [1], p. 434.
[7] See [1], pp. 433–436.
[8] [1], p. 433.

holds. Also, again by the Assignment Axiom and the Consequence Rule, we have that

$$\{x \geq 0 \wedge a = x!/x!\}\ b := x\ \{a = x!/b!\}.$$

Hence, using the Composition Rule, it follows that (1) holds, *i.e.* that

$$\{x \geq 0\}\ a := 1; b := x\ \{a = x!/b!\}$$

holds. Finally, consider (2): we can use the Assignment Axiom and Consequence Rule to conclude both

$$\{ab = x!/(b-1)!\}\ a := ab\ \{a = x!/(b-1)!\}$$

and

$$\{a = x!/(b-1)!\}\ b := b-1\ \{a = x!/b!\},$$

from which it follows by the Composition Rule that

$$\{ab = x!/(b-1)!\}\ a := ab; b := b-1\ \{a = x!/b!\}$$

holds, from which it follows by the Consequence Rule that

$$\{a = x!/b! \wedge b \neq 0\}\ a := ab; b := b-1\ \{a = x!/b!\}$$

holds, which is to say that (2) holds, as required. ∎

Let's refer to the program in the preceding example as the *Factorial Program*. For the sake of the exposition, I will use it to illustrate the alternative semantic treatments of **while** presented here.[9]

## 3.2 The axiom of invariance for while

The extended trace language permits the formulation of a **while** axiom based on the idea of an invariant:

$$[\mathrm{acc}(T,S,T') \wedge \neg V(T',\phi)] \rightarrow .\forall R\,[[V(R,p\wedge\phi)\wedge \mathrm{ext}(\phi,T,S,R)] \rightarrow V(R.S,p)] \rightarrow .V(T,p) \rightarrow V(T.\ \mathbf{while}\,\phi\,\mathbf{do}\,S\,\mathbf{od}\,.\,p\wedge\neg\phi)$$

This axiom states that *if* the truth of the invariant $p$ is preserved whenever $S$ is executed on a trace in which both $p$ and the loop condition $\phi$ hold and which is an $S$-accessible $\phi$-extension of a trace $T$, *then* if $p$ is true at $T$, then $p$ is also true at $T.\ \mathbf{while}\,\phi\,\mathbf{do}\,S\,\mathbf{od}$, unless this loop does not terminate.

Now let's prove the Factorial Program correct using the extended trace language (as modified to include the axiom of invariance instead of the recursive **while** axiom). We wish to derive:

**Proposition 2** $\exists T'\,[\mathrm{acc}(a := 1.b := x, a := ab.b := b-1, T') \wedge \neg V(T', b \neq 0)] \rightarrow .x \geq 0 \rightarrow .V(a := 1.b := x.\ \mathbf{while}\ b \neq 0\ \mathbf{do}\ a := ab.b := b-1\ \mathbf{od}\,.\,a = x!)$.

**Proof:** In order to use the axiom of invariance, we need two lemmas. (We use the same invariant as was used above in the Hoare logic correctness proof.) One of these lemmas is trivial, namely

**(L1)** $x \geq 0 \rightarrow V(a := 1.b := x, a = x!/b!)$,

[9] I will not here present an extended trace language correctness proof of the Factorial Program using the recursive axiom for **while**, since McLean does this (using essentially the same specification) in [6].

which follows from extended trace language axioms 3, 4, and 5. The other lemma is this:

(L2) $x \geq 0 \rightarrow .\forall R\,[[V(R, a = x!/b! \wedge b \neq 0) \wedge \mathrm{ext}(b \neq 0, a := 1; b := x, a := ab; b := b - 1, R)] \rightarrow V(R.a := ab.b := b - 1, a = x!/b!)]$.

Let $R$ be given such that $\mathrm{ext}(b \neq 0, a := 1.b := x, a := ab.b := b - 1, R)$. It follows that

$$V(R, a = x!/b! \wedge b \neq 0) \Rightarrow V(R, ab = x!/(b-1)! \wedge b \neq 0)$$
$$\Rightarrow V(R.a := ab, a = x!/(b-1)! \wedge b \geq 1) \text{ by axioms 3,4,5;}$$
$$\Rightarrow V(R.a := ab.b := b - 1, a = x!/b)! \wedge b \geq 1) \text{ by axioms 3,4,5.}$$

This completes the proof of (L2). (L1) and (L2), together with our hypothesis and the invariance axiom imply:

$\exists T'\,[\mathrm{acc}(a := 1.b := x, a := ab.b := b - 1, T') \wedge \neg V(T', b \neq 0)] \rightarrow .x \geq 0 \rightarrow .V(a := 1; b := x;$ **while** $b \neq 0$ **do** $a := ab.b := b - 1$ **od**$, a = x!/b \wedge b = 0)$,

which by arithmetic implies the desired result, namely:

$\exists T'\,[\mathrm{acc}(a := 1.b := x, a := ab.b := b - 1, T') \wedge \neg V(T', b \neq 0)] \rightarrow .x \geq 0 \rightarrow .V(a := 1; b := x;$ **while** $b \neq 0$ **do** $a := ab.b := b - 1$ **od**$, a = x!)$. ∎

Notice that, unlike the recursive semantics for **while**, this invariance-based semantic treatment of **while** does not seem to require us to use mathematical induction to prove the correctness of the Factorial Program.[10] Like all good things, however, this feature must be paid for: we will use induction to prove that the recursive axiom implies the invariance axiom (see Lemma 2 in section 5 below).

# 4 The precise count while axiom

In this section we consider a third and somewhat different semantics for **while**.

$$[\neg V(T.S^n, \phi) \wedge \forall k\,[0 \leq k < n \rightarrow V(T.S^k, \phi)]] \rightarrow V(T.\,\textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\,.\,x) = V(T.S^n.x)$$

That is, if, starting at $T$, $\phi$ first becomes false after $n$ iterations of $S$, then $T.S^n$ is simply equivalent to $T.$ **while** $\phi$ **do** $S$ **od**. Like the invariance axiom (and unlike the recursive axiom) the precise count axiom has the following property: one must choose carefully which of its instances to use. In particular, one must find or correctly guess how many times the loop will iterate in order to use this axiom. In some cases where this is easy to guess, this axiom could make the proof somewhat simpler. For illustration, consider once again the Factorial Program. We wish to prove:

**Proposition 3** $\exists T'\,[\mathrm{acc}(a := 1.b := x, a := ab.b := b - 1, T') \wedge \neg V(T', b \neq 0)] \rightarrow .x \geq 0 \rightarrow .V(a := 1; b := x;$ **while** $b \neq 0$ **do** $a := ab.b := b - 1$ **od**$, a) = x!$.

**Proof:** Let $\exists T'\,[\mathrm{acc}(a := 1.b := x, a := ab.b := b - 1, T')$, $\neg V(T', b \neq 0)$, and $x \geq 0$ all hold. Let $R$ be the shortest $T'$ satisfying the first assumption, and let $n$ be the number of iterations of the loop $a := ab.b := b - 1$ in $R$. We will show that $n = x$, *i.e.* that $x$ satisfies the hypothesis of the precise count axiom.

[10] I conjecture that the recursive semantics in [6] does in fact require the use of induction. An examination of McLean's correctness proof on page 7 of [6] provides some evidence for this conjecture.

First we need to show that $\neg V(a := 1.b := x.(a := ab.b := b - 1)^x, b \neq 0)$, *i.e.* $V(a := 1.b := x.(a := ab.b := b - 1)^x, b) = 0$. We show this by giving a proof by induction of the following identity:

**(L3)** $V(a := 1.b := x.(a := ab.b := b - 1)^k, b) = x - k$.

**Proof-of-(L3):** Since $V(a := 1.b := x, b) = x$, we have the basis case. Suppose that the identity holds for $k \leq m$, and let $k = m + 1$. Then

$$\begin{aligned} V(a := 1.b := x.(a := ab.b := b - 1)^{m+1}, b) &= V(a := 1.b := x.(a := ab.b := b - 1)^m.a := ab.b - 1) \text{ by axiom 3;} \\ &= V(a := 1.b := x.(a := ab.b := b - 1)^m, b - 1) \text{ by axiom 6;} \\ &= V(a := 1.b := x.(a := ab.b := b - 1)^m, b) - 1 \text{ by axiom 4;} \\ &= (x - m) - 1 \text{ by inductive hypothesis;} \\ &= x - (m + 1) \text{ as required.} \end{aligned}$$

This completes the proof of (L3). Note that (L3) implies:

$$V(a := 1.b := x.(a := ab.b := b - 1)^x, b) = x - x = 0,$$

which establishes the first conjunct of the antecedent of the precise count axiom. To establish the second conjunct, we need to show:

$$\forall k[0 \leq k < x \rightarrow V(a := 1.b := x.(a := ab.b := b - 1)^k, b \neq 0)].$$

But this also follows from the (L3), since $k < x$ implies $x - k \neq 0$.

Having established both conjuncts of the antecedent of the precise count axiom we now must show that the following holds:

**(L4)** $0 \leq y \leq x \rightarrow V(a := 1.b := x.(a := ab.b := b - 1)^y, a) = x!/(x - y)!$.

**Proof-of-(L4):** Again we proceed by induction. Assume $0 \leq y \leq x$. Clearly $V(a := 1.b := x, a) = 1 = x!/(x - 0)!$, so the basis case holds. Now suppose (L4) holds for $y \leq k$, and consider the case where $y = k+1$. We reason as follows: we have $V(a := 1.b := x.(a := ab.b := b - 1)^k, a) = x!/(x - k)!$ by the inductive hypothesis, and $V(a := 1.b := x.(a := ab.b := b - 1)^k, b) = x - k$ by (L3). Thus $V(a := 1.b := x.(a := ab.b := b - 1)^k, a = x!/(x - k)!)$, which in turn implies $V(a := 1.b := x.(a := ab.b := b - 1)^k, ab = [x!/(x - k)!](x - k))$ by axioms 4 and 5. This implies that $V(a := 1.b := x.(a := ab.b := b - 1)^k, ab = x!/(x - (k + 1)!))$ holds, since $x > k$, which holds because $x \geq y = k + 1$. Thus $V(a := 1.b := x.(a := ab.b := b - 1)^k, ab) = x!/(x - (k + 1))!$ holds, which implies by axiom 3 that $V(a := 1.b := x.(a := ab.b := b - 1)^k.a := ab, a) = x!/(x - (k + 1))!$ holds. Consequently we have

$$\begin{aligned} V(a := 1.b := x.(a := ab.b := b - 1)^{k+1}, a) &= V(a := 1.b := x.(a := ab.b := b - 1)^k.a := ab.b := b - 1, a) \\ &= V(a := 1.b := x.(a := ab.b := b - 1)^k.a := ab, a) \text{ by axiom 6;} \\ &= x!/(x - (k + 1))!, \end{aligned}$$

as required. This completes the proof of (L4). Now, at last, we can derive our main conclusion using the precise count axiom itself.

$$\begin{aligned} V(a := 1.b := x. \textbf{ while } b \neq 0 \textbf{ do } a := ab.b := b - 1 \textbf{ od}, a) &= V(a := 1.b := x.(a := ab.b := b - 1)^x, a) \\ &= x! \text{ by (L4)} \end{aligned}$$

This completes our correctness proof for the Factorial Program using the precise count axiom. ∎

## 5 Equivalence of the three axioms

As one would hope, we can prove that our three trace axioms for the **while** construct are equivalent. The three axioms, once again, are as follows:

(a) $[\neg V(T.S^n, \phi) \wedge \forall k[0 \leq k < n \rightarrow V(T.S^k, \phi)] \rightarrow V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ , x) = V(T.S^n, x)$

(b) $(\mathrm{acc}(S, S_0, T') \wedge \neg V(T', \phi)) \rightarrow V(S.\ \textbf{while}\ \phi\ \textbf{do}\ S_0\ \textbf{od}\ , x) = V(S.\ \textbf{if}\ \phi\ \textbf{then}\ S_0.\ \textbf{while}\ \phi\ \textbf{do}\ S_0\ \textbf{od}\ \textbf{fi}\ , x)$

(c) $[\mathrm{acc}(T, S, T') \wedge \neg V(T', \phi)] \rightarrow .\forall R\left[[V(R, p \wedge \phi) \wedge \mathrm{ext}(\phi, T, S, R)] \rightarrow V(R.S, p)\right] \rightarrow .V(T, p) \rightarrow V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ , p \wedge \neg\phi)$

where $x$ is a program variable or Boolean expression, with '=' understood as meaning $\leftrightarrow$ in the latter case. The first lemma we need is:

**Lemma 1** *Axiom (a), the precise count axiom, implies Axiom (b), the recursive axiom.*

**Proof:** Suppose that (a) holds and suppose that $\mathrm{acc}(S, S_0, T') \wedge \neg V(T', \phi)$ holds. By the definition of acc, let $n$ be the smallest $n'$ such that $n' \geq 0 \wedge \neg V(T.S^{n'}, \phi)$ holds. Hence $V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ , x) = V(T.S^n, x)$. There are two cases.

*Case 1*: If $n = 0$, then $\neg V(T, \phi)$ holds, and we also have

$$\begin{aligned} V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ , x) &= V(T, x) \text{ by Axiom (a);} \\ &= V(\ \textbf{if}\ \phi\ \textbf{then}\ S.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ \textbf{fi}\ , x) \text{ by Axiom 10;} \end{aligned}$$

*Case 2*: If $n > 0$, then we have $V(T, \phi)$; note that $n - 1$ is the smallest $n'$ such that $n' \geq 0 \wedge \neg V(T.S.S^{n'}, \phi)$ holds. This gives us

$$\begin{aligned} V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ , x) &= V(T.S.S^{n-1}, x) \text{ by Axiom (a);} \\ &= V(T.S.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od} \text{ by Axiom (a);} \\ &= V(T.\ \textbf{if}\ \phi\ \textbf{then}\ S.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ \textbf{fi}\ , x) \text{ by Axiom 9.} \end{aligned}$$

So in either case, $V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ , x) = V(T.\ \textbf{if}\ \phi\ \textbf{then}\ S.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ \textbf{fi}\ , x)$, as required. ∎

**Lemma 2** *Axiom (b) implies Axiom (c), the invariance axiom.*

**Proof:** Suppose that (b) holds and suppose that

(i) $\mathrm{acc}(T, S, T') \wedge \neg V(T', \phi)$

(ii) $\forall R\left[[V(R, p \wedge \phi) \wedge \mathrm{ext}(\phi, T, S, R)] \rightarrow V(R.S, p)\right]$, and

(iii) $V(T, p)$

also hold. We need to show that $V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ , p \wedge \neg\phi)$ holds, as well. By the definition of acc we know that $\exists n'\,[n' \geq 0 \wedge \neg V(T.S^{n'}, \phi)]$. Let $n$ be the smallest such $n'$. The proof will be by induction on the value of $n$.

*Basis Case*: Let $n = 0$. Hence $\neg V(T, \phi)$ holds, and so

$$\begin{aligned} V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ , p \wedge \neg\phi) &\Longleftrightarrow V(T.\ \textbf{if}\ \phi\ \textbf{then}\ S.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ \textbf{fi}\ , p \wedge \neg\phi \text{ by (a);} \\ &\Longleftrightarrow V(T.skip, p \wedge \neg\phi) \text{ by axiom 10;} \\ &\Longleftrightarrow V(T, p \wedge \neg\phi) \text{ by axiom 12,} \end{aligned}$$

but $V(T, p \wedge \neg\phi)$ is true by hypothesis, so $V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ , p \wedge \neg\phi)$ is true, as required.

*Inductive Step*: Suppose that Axiom (b) implies Axiom (c) for all values $k$ such that $0 \leq k \leq n$, where $n > 0$. We show that this implication holds for $n+1$, as well.

Since $n > 0$, we know that $V(T, \phi)$ holds, which together with (iii) implies that $V(T, p \wedge \phi)$ holds. Since, in addition, we know that $\mathrm{acc}(T, S, T)$, it follows by (ii) that $V(T.S, \phi)$ holds. Since $n > 0$ we also know that $\mathrm{acc}(T.S, S, T')$ holds, too. In particular, $n$ is the smallest $n'$ such that $n' \geq 0 \wedge \neg V(T.S.S^{n'}, \phi)$

Note that (ii) implies $\forall R\,[[V(R, p \wedge \phi) \wedge \mathrm{ext}(\phi, T.S, S, R)] \rightarrow V(R.S, p)]$. Since $V(T, \phi)$ and $V(T, p)$ both hold, it also follows from (ii) that $V(T.S, p)$ holds.

Thus we have

$$V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}, p \wedge \neg\phi) \Longleftrightarrow V(T.\ \textbf{if}\ \phi\ \textbf{then}\ S.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}\ \textbf{fi}, p \wedge \neg\phi) \text{ by (a);}$$
$$\Longleftrightarrow V(T.S.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}, p \wedge \neg\phi) \text{ by axiom 9;}$$

but, by the inductive hypothesis, $V(T.S.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}, p \wedge \neg\phi)$ is true, since $\mathrm{acc}(T.S, S, T') \wedge \neg V(T', \phi)$, $\forall R\,[[V(R, p \wedge \phi) \wedge \mathrm{ext}(\phi, T.S, S, R)] \rightarrow V(R.S, p)]$, and $V(T.S, p)$ all hold. Hence

$$V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}, p \wedge \neg\phi)$$

holds as required. ∎

**Lemma 3** *Axiom (c) implies Axiom (a)*

**Proof:** Suppose that (c) holds, and let $n$ be such that $\neg V(T.S^n, \phi) \wedge \forall k\,[0 \leq k < n \rightarrow V(T.S^k, \phi)]$ holds. Let $y = V(T.S^n, x)$, and let $p$ be the formula $\phi \vee x = y$. We will show that $V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}, x) = y$ by first showing $V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}, p \wedge \neg\phi)$, and to show this it suffices (by axiom (c)) to show that (i) $[\mathrm{acc}(T, S, T') \wedge \neg V(T', \phi)]$, (ii) $\forall R\,[[V(R, (\phi \vee x = y) \wedge \phi) \wedge \mathrm{ext}(\phi, T, S, R)] \rightarrow V(R.S, (\phi \vee x = y))]$, and (iii) $V(T, (\phi \vee x = y))$. Now, (i) and (iii) follow straightforwardly from our hypothesis. Let us then examine (ii).

Let $R$ be such that $V(R, (\phi \vee x = y) \wedge \phi) \wedge \mathrm{ext}(\phi, T, S, R)$. Since $\phi$ entails $(\phi \vee x = y)$, this is equivalent to $V(R, \phi) \wedge \mathrm{ext}(\phi, T, S, R)$. It follows that $R$ has the form $T.S^k$ for some $k$ such that $0 \leq k < n$, since otherwise (by hypothesis) $\phi$ would not be uniformly true over the $S$-extensions of $T$ that are prefixes of $R$. There are two cases:

*Case 1*: If $k = n - 1$, then by hypothesis

$$V(R.S, (\phi \vee x = y)) \Longleftrightarrow V(T.S^{n-1}.S, (\phi \vee x = y))$$
$$\Longleftrightarrow V(T.S^n, (\phi \vee x = y)),$$

but $V(T.S^n, (\phi \vee x = y))$ is true, since $V(T.S^n, x = y)$ holds, and so $V(R.S, (\phi \vee x = y))$ is true, as well.

*Case 2*: If $k < n - 1$, then $V(T.S^{k+1}, \phi)$ holds, since $k + 1 < n$, and

$$V(R.S, (\phi \vee x = y)) = V(T.S^k.S, (\phi \vee x = y))$$
$$= V(T.S^{k+1}, (\phi \vee x = y)),$$

but $V(T.S^{k+1}, (\phi \vee x = y))$ is true, since $V(T.S^{k+1}, \phi)$ holds, and so $V(R.S, (\phi \vee x = y))$ is true, as well.

This suffices to establish (ii), and hence, by (c), (i), (ii), and (iii) we have $V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}, (\phi \vee x = y) \wedge \neg\phi)$, which implies $V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}, x = y)$, which implies $V(T.\ \textbf{while}\ \phi\ \textbf{do}\ S\ \textbf{od}, x) = V(T.S^n, x)$, as required. ∎

**Theorem 1** *Axioms (a), (b), and (c) are equivalent.*

**Proof:** By Lemmas 1, 2, 3. ∎

## Acknowledgement

I am grateful to John McLean for reading and providing comments on the penultimate draft of this report.

## References

[1] Apt, Krzysztof R., "Ten Years of Hoare's Logic: A Survey," *ACM Transactions on Programming Languages and Systems* 3: 431–483, October 1981.

[2] de Bakker, Jaco, *Mathematical Theory of Program Correctness*, Englewood Cliffs: Prentice-Hall, 1980.

[3] Gabbay, Dov and Franz Guenthner (eds.), *Handbook of Philosophical Logic*, vol. 2. Dordrecht: D. Reidel, 1984.

[4] Harel, David, "Dynamic Logic," in [3], pp. 497–604.

[5] McLean, John, "A Formal Method for the Abstract Specification of Software," *Journal of the ACM* 31: 600–627, July 1984.

[6] McLean, John, "Using Trace Specifications for Program Semantics and Verification," NRL Report 9033, Naval Research Laboratory, Washington, D.C., 1987.